

Predicting Daily Land Average Temperature

Load the Data

Load the daily land-surface average anomaly data provided by Berkeley Earth collected from 1880 to 2022.

```
In [1]: import pandas as pd

url = "https://berkeley-earth-temperature.s3.us-west-1.amazonaws.com/Global/
...
read the data from the url link
ignore the comments starting with '%'
ignore the header in the comments and assign manually
...
df = pd.read_csv(url, sep=r"\s+", comment="%", header=None)

# assign column headers
column_names = ["Date Number", "Year", "Month", "Day", "Day of Year", "Anomaly"]
df.columns = column_names

# df.to_csv("../data/raw.csv", index=False)
```

```
In [2]: df
```

```
Out[2]:
```

	Date Number	Year	Month	Day	Day of Year	Anomaly
0	1880.001	1880	1	1	1	-0.692
1	1880.004	1880	1	2	2	-0.592
2	1880.007	1880	1	3	3	-0.673
3	1880.010	1880	1	4	4	-0.615
4	1880.012	1880	1	5	5	-0.681
...
52072	2022.568	2022	7	27	208	1.639
52073	2022.571	2022	7	28	209	1.631
52074	2022.574	2022	7	29	210	1.574
52075	2022.577	2022	7	30	211	1.577
52076	2022.579	2022	7	31	212	1.629

52077 rows × 6 columns

Data Preprocessing

```
In [3]: df.isna().sum()
```

```
Out[3]: Date Number    0
        Year          0
        Month          0
        Day            0
        Day of Year    0
        Anomaly        0
        dtype: int64
```

```
In [4]: df.dtypes
```

```
Out[4]: Date Number    float64
        Year          int64
        Month          int64
        Day            int64
        Day of Year    int64
        Anomaly        float64
        dtype: object
```

```
In [5]: # df = df.drop(columns=['Date Number'])
```

```
In [6]: BASELINE_TEMP = 8.59 # Jan 1951–Dec 1980 land-average temperature in celsius
        df['Temperature'] = df['Anomaly'] + BASELINE_TEMP
```

```
In [7]: month_dict = {
        1: 'January',
        2: 'February',
        3: 'March',
        4: 'April',
        5: 'May',
        6: 'June',
        7: 'July',
        8: 'August',
        9: 'September',
        10: 'October',
        11: 'November',
        12: 'December'
    }

    df['Month_Name'] = df['Month'].map(month_dict)
```

```
In [8]: df
```

Out [8]:

	Date Number	Year	Month	Day	Day of Year	Anomaly	Temperature	Month_Name
0	1880.001	1880	1	1	1	-0.692	7.898	January
1	1880.004	1880	1	2	2	-0.592	7.998	January
2	1880.007	1880	1	3	3	-0.673	7.917	January
3	1880.010	1880	1	4	4	-0.615	7.975	January
4	1880.012	1880	1	5	5	-0.681	7.909	January
...
52072	2022.568	2022	7	27	208	1.639	10.229	July
52073	2022.571	2022	7	28	209	1.631	10.221	July
52074	2022.574	2022	7	29	210	1.574	10.164	July
52075	2022.577	2022	7	30	211	1.577	10.167	July
52076	2022.579	2022	7	31	212	1.629	10.219	July

52077 rows x 8 columns

Exploratory Data Analysis (EDA)

The goal of this EDA is to determine what type of predictive model will be the best fit for the data. A suitable regression method is to be explored, as the target (global average land temperature) is continuous. After preprocessing the dataset, no presence of null values were found requiring attention. In order to make the Month_Name (converting from Month) feature more readable during EDA, the feature was converted to an ordinal feature, with Jan as the first in the order and December as the last in the order. However, monthly trends in the data were not considered as the measurements for temperature are taken globally, where temperature varies seasonally in different locations.

Nonetheless, during the initial EDA, this ordinal conversion was helpful when visualizing the data with respect to the month of data collection. A cutoff year was decided for splitting the data into test and training data sets (see code below), and EDA was carried out on only the training portion of the data set to avoid violating the golden rule and double dipping. A randomized test split was not implemented as the model is desired for predicting temperatures in the future, so test data taken from the most recent years represents the best evaluation of the regression model. Currently the test data includes the most recent five years of data (2022, 2021, 2020, 2019 and 2018) whereas the rest of the data was allocated to the training data set, reaching as far back as 1880. Once the dataset was split, EDA was performed with several visualizations the most informative of which are presented below. Discussions of the findings and rationale are found below as well.

```
In [9]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52077 entries, 0 to 52076
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Date Number     52077 non-null  float64
1   Year            52077 non-null  int64
2   Month           52077 non-null  int64
3   Day             52077 non-null  int64
4   Day of Year     52077 non-null  int64
5   Anomaly         52077 non-null  float64
6   Temperature     52077 non-null  float64
7   Month_Name      52077 non-null  object
dtypes: float64(3), int64(4), object(1)
memory usage: 3.2+ MB
```

```
In [10]: df.describe()
```

```
Out[10]:
```

	Date Number	Year	Month	Day	Day of Year	
count	52077.000000	52077.000000	52077.000000	52077.000000	52077.000000	52077
mean	1951.290840	1950.791693	6.512779	15.729228	182.688577	182.688577
std	41.160006	41.160470	3.447762	8.799991	105.302088	105.302088
min	1880.001000	1880.000000	1.000000	1.000000	1.000000	1.000000
25%	1915.648000	1915.000000	4.000000	8.000000	92.000000	92.000000
50%	1951.292000	1951.000000	7.000000	16.000000	183.000000	183.000000
75%	1986.936000	1986.000000	10.000000	23.000000	274.000000	274.000000
max	2022.579000	2022.000000	12.000000	31.000000	365.000000	365.000000

```
In [11]: df.isna().any()
```

```
Out[11]: Date Number    False
Year                False
Month               False
Day                 False
Day of Year         False
Anomaly             False
Temperature         False
Month_Name          False
dtype: bool
```

```
In [12]: # month order from above dict values
month_order = list(month_dict.values())
month_order
```

```
Out[12]: ['January',
          'February',
          'March',
          'April',
          'May',
          'June',
          'July',
          'August',
          'September',
          'October',
          'November',
          'December']
```

```
In [13]: # Order the month names as categorical
df['Month_Name'] = pd.Categorical(df['Month_Name'], categories=month_order,
df['Month_Name']
```

```
Out[13]: 0      January
1      January
2      January
3      January
4      January
...
52072   July
52073   July
52074   July
52075   July
52076   July
Name: Month_Name, Length: 52077, dtype: category
Categories (12, object): ['January' < 'February' < 'March' < 'April' ... 'S
eptember' < 'October' < 'November' < 'December']
```

```
In [14]: # Split into test and train dataframes based on cutoff
# beyond which we would like to score our model and use on unseen examples c
cutoff = 2018
test_df = df[df['Year']>=cutoff]
test_df
```

Out[14]:

	Date Number	Year	Month	Day	Day of Year	Anomaly	Temperature	Month_Name
50404	2018.001	2018	1	1	1	1.194	9.784	January
50405	2018.004	2018	1	2	2	1.314	9.904	January
50406	2018.007	2018	1	3	3	1.358	9.948	January
50407	2018.010	2018	1	4	4	1.390	9.980	January
50408	2018.012	2018	1	5	5	1.517	10.107	January
...
52072	2022.568	2022	7	27	208	1.639	10.229	July
52073	2022.571	2022	7	28	209	1.631	10.221	July
52074	2022.574	2022	7	29	210	1.574	10.164	July
52075	2022.577	2022	7	30	211	1.577	10.167	July
52076	2022.579	2022	7	31	212	1.629	10.219	July

1673 rows × 8 columns

In [15]: `train_df = df[df['Year']<cutoff]`
`train_df`

Out[15]:

	Date Number	Year	Month	Day	Day of Year	Anomaly	Temperature	Month_Name
0	1880.001	1880	1	1	1	-0.692	7.898	January
1	1880.004	1880	1	2	2	-0.592	7.998	January
2	1880.007	1880	1	3	3	-0.673	7.917	January
3	1880.010	1880	1	4	4	-0.615	7.975	January
4	1880.012	1880	1	5	5	-0.681	7.909	January
...
50399	2017.988	2017	12	27	361	1.046	9.636	December
50400	2017.990	2017	12	28	362	1.445	10.035	December
50401	2017.993	2017	12	29	363	1.594	10.184	December
50402	2017.996	2017	12	30	364	1.506	10.096	December
50403	2017.999	2017	12	31	365	1.302	9.892	December

50404 rows × 8 columns

```
In [16]: import altair as alt

# Simplify Working with Large Datasets
alt.data_transformers.enable('vegafusion')

# Ignore all warnings from the altair package for pdf rendering, warnings va
# resource for implementation:
# https://stackoverflow.com/questions/3920502/how-to-suppress-a-third-party-
import warnings
warnings.filterwarnings('ignore', module='altair')

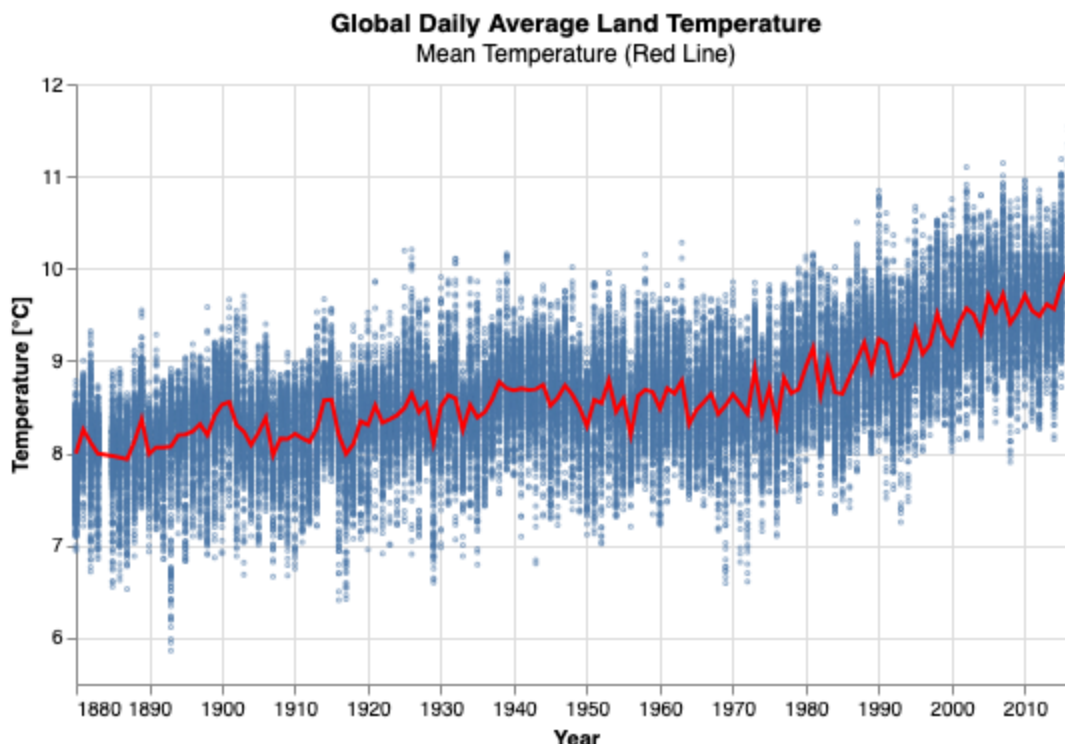
# Configure Plot Sizes for pdf
plot_size = {'width': 500, 'height': 300}
```

Due to the many measurements that occur in each year (contributing to plotting noise), the mean of the temperatures for each group of measurements taken in a year were plotted in addition to the raw data points of temperature over time. A general trend of increasing temperature over time with local fluctuations can be observed below.

```
In [17]: temp_points = alt.Chart(train_df,
    title=alt.Title(
        text='Global Daily Average Land Temperature',
        subtitle='Mean Temperature (Red Line)')
    ).mark_point(opacity=0.6, size=1).encode(
        x = 'Year:T',
        y = 'Temperature:Q'
    )
temp_line_mean = temp_points.mark_line(size=2, color='red').encode(
    x = alt.X('Year:T', title='Year'),
    y = alt.Y('mean(Temperature):Q', title='Temperature [°C]'
        ).scale(zero=False)
).properties(**plot_size)

temp_points+temp_line_mean
```

Out [17]:

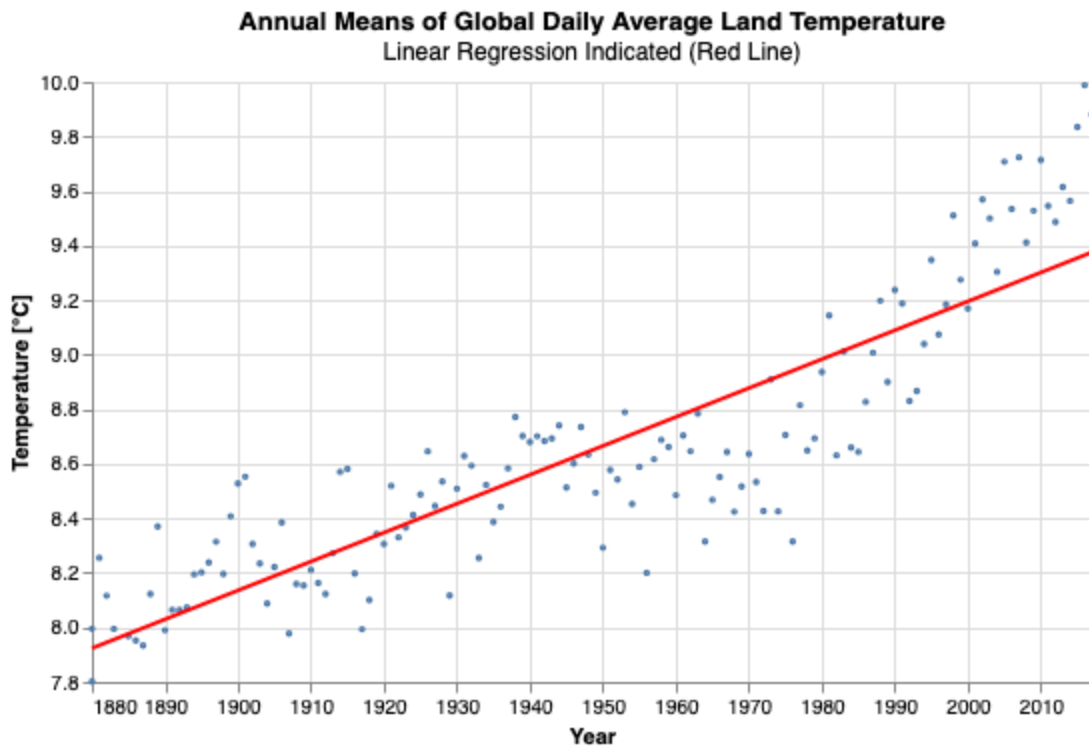


Using the mean data points by year, a linear regression model was fit to the training data to assess the viability of this approach in prediction. The linear regression fit can be seen to capture the overall trend well, while missing local fluctuations above and below the line. This indicates that a linear model has potential to generalize well unseen year examples, but will not be precise in its predictions for every year.

```
In [18]: temp_points_avg = alt.Chart(train_df,
    title=alt.Title(
        text='Annual Means of Global Daily Average Land Temperature',
        subtitle='Linear Regression Indicated (Red Line)')
    ).mark_point(size=2).encode(
    x = alt.X('Year:T', title='Year'),
    y = alt.Y('mean(Temperature):Q',
        title='Temperature [°C]').scale(zero=False)
    )

reg = temp_points_avg+temp_points_avg.mark_line(
    size=2, color='red').transform_regression(
    'Year',
    'Temperature'
).properties(**plot_size)
reg
```

Out [18]:



All 5-year rolling average was explored as an option as well, due to its ability to adapt the curve more effectively to local fluctuations. However, a model of this type may be overcomplicated for our use case and is difficult to accurately extrapolate the curve to predict future temperatures.

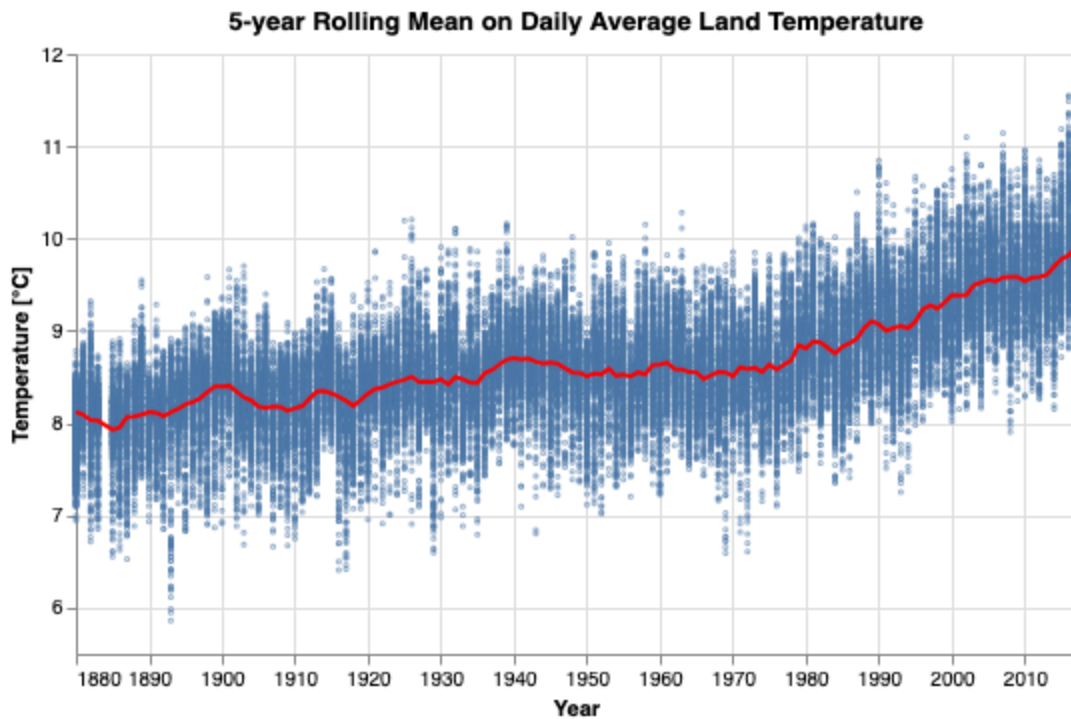
```
In [19]: # 30-day rolling average example adapted from source documentation below:
# https://altair-viz.github.io/gallery/scatter_with_rolling_mean.html
# # Adaptations from 531 Lecture 5 Notes

mean_per_year = train_df.groupby(['Year'])['Temperature'].mean().reset_index

roll_line = alt.Chart(mean_per_year).mark_line(
    color='red',
    size=2
).transform_window(
    rolling_mean='mean(Temperature)',
    frame=[-2, 2]
).encode(
    x='Year:T',
    y=alt.Y('rolling_mean:Q',
            title='Temperature [°C]').scale(zero=False)
)

# Five-year moving/rolling average
(temp_points+roll_line).properties(**plot_size,
    title = '5-year Rolling Mean on Daily Average Land Temperature')
```

Out [19]:



Finally, the distribution of the training data for temperature with respect to time was evaluated for distinct eras in the box plot below. The most recent year plotted is one year before the cutoff year defined above. We can see from the boxplot that for each era, the mean (black point) is close to the median (white horizontal line), suggesting that there is low skewness in the target. It is also apparent that each era has a few outliers far away from the mean and median, which may be important for the model and hyperparameters. Ideally the model will not treat outliers with too much weighting, as they are infrequent but large in magnitude. The vast majority of the data (from a qualitative viewpoint) resides between the upper and lower quartiles, as seen by the short box lengths relative to the range of the target. Overall, a linear regression model seems to be a feasible approach based on the EDA.

```
In [20]: years_selection = [1880, 1920, 1960, 2000, cutoff-1]

box = alt.Chart(train_df[train_df['Year'].isin(years_selection)]),
        title=alt.Title(
            text='Global Daily Average Land Temperature Distributions by Select
            subtitle='Annual Means Indicated by Black Points'))
        .mark_boxplot()
        .encode(
            x = 'Year:N',
            y = alt.Y('Temperature').scale(zero=False),
            color = 'Year:N'
        )
box_point = alt.Chart(train_df[train_df['Year'].isin(years_selection)]).mark
        x = 'Year:N',
        y = alt.Y('mean(Temperature)').scale(zero=False),
        color = alt.value('black')
    )
```

```
comb = box+box_point
comb.properties(**plot_size)
```

Out[20]:

